# Homework Assignment 1

**Due: January 27, 2017 at 23:00**

| | |
|---|---|
| **Total points:** | 100 |
| **Deliverables:** | hw1.pdf containing typeset solutions to Problems 1-12. |
| | k-coloring.rkt containing your implementation for Problem 10. |
| **Sources:** | https://gitlab.cs.washington.edu/cse507/hw17wi. |

## 1 Propositional Logic and Normal Forms (30 points)

1. (2 points) Decide whether each of the following formulas is valid. If the formula is valid, prove its validity using the semantic argument method. Otherwise, provide a falsifying interpretation and, if the formula is satisfiable, a satisfying interpretation.

   (a) $(p \land q) \to (p \to q)$

   (b) $(p \to (q \to r)) \to (\neg r \to (\neg q \to \neg p))$

   (**LaTeX Hint:** use the mathpartir package to typeset proofs.)

2. (3 points) Convert the following formula to equivalent formulas in NNF, CNF, and DNF. Write the final formula as the answer; the intermediate conversion steps need not be shown.

$$\neg(\neg(p \land q) \to \neg r)$$

3. (5 points) Convert the formula from Problem 2 to an equisatisfiable formula in CNF using Tseitin's encoding. Write the final CNF formula as the answer. Use $a_\phi$ to denote the auxiliary variable for the formula $\phi$; for example, $a_{(p \land q)}$ should be used to denote the auxiliary variable for $(p \land q)$.

4. (10 points) Let $\phi$ be a propositional formula in NNF, and let $I$ be an interpretation of $\phi$. Let the *positive set* of $I$ with respect to $\phi$, denoted $pos(I, \phi)$, be the literals of $\phi$ that are satisfied by $I$. As an example, for the NNF formula $\phi = (\neg r \land p) \lor q$ and the interpretation $I = [r \mapsto \bot, p \mapsto \top, q \mapsto \bot]$, we have $pos(I, \phi) = \{\neg r, p\}$. Prove the following theorem about the monotonicity of NNF:

   **Monotonicity of NNF:** For every interpretation $I$ and $I'$ such that $pos(I, \phi) \subseteq pos(I', \phi)$, if $I \models \phi$, then $I' \models \phi$.

   (**Hint:** Use structural induction.)

5. (10 points) Let $\phi$ be an NNF formula. Let $\hat{\phi}$ be a formula derived from $\phi$ using a modified version of Tseitin's encoding in which the CNF constraints are derived from implications rather than bi-implications. For example, given a formula

$$a_1 \land (a_2 \lor \neg a_3),$$

   the new encoding is the CNF equivalent of the following formula, where $x_0, x_1, x_2$ are fresh auxiliary variables:

$$\begin{array}{ll} x_0 & \land \\ (x_0 \to a_1 \land x_1) & \land \\ (x_1 \to a_2 \lor x_2) & \land \\ (x_2 \to \neg a_3) & \end{array}$$

Note that Tseitin's encoding to CNF starts with the same formula, except that $\rightarrow$ is replaced with $\leftrightarrow$. As a result, the new encoding has roughly half as many clauses as the Tseitin's encoding.

Prove that $\hat{\phi}$ is satisfiable if and only if $\phi$ is satisfiable.

(**Hint**: Use the theorem from Problem 4.)

# 2 SAT solving (10 points)

6. (10 points) Consider the following set of clauses:

$$
\begin{aligned}
c_1 &: (\neg x_1 \lor x_2 \lor x_4) \\
c_2 &: (x_1 \lor x_3) \\
c_3 &: (\neg x_4 \lor \neg x_2) \\
c_4 &: (\neg x_4 \lor \neg x_1 \lor x_2) \\
c_5 &: (x_3 \lor \neg x_1) \\
c_6 &: (\neg x_3 \lor \neg x_2 \lor x_4) \\
c_7 &: (x_1 \lor x_4) \\
c_8 &: (\neg x_2 \lor x_1)
\end{aligned}
$$

Complete the table below to show how a modern CDCL SAT solver (Lecture 2) decides the satisfiability of these clauses. Start a new row every time the decision level changes (due to a new decision or backtracking). Show the implication graph at each decision level by listing all of its labeled edges. Keep all conflict clauses returned by AnalyzeConflict and assign them names $c_9$, $c_{10}$, etc.

Use the following rules when making choices while executing the CDCL algorithm:

- For choosing the next assignment in the Decide step, use the Dynamic Largest Individual Sum (DLIS) heuristic. In the case of a tie between variables, pick the $x_i$ with the lowest index $i$. If there is a tie between $x_i$ and $\neg x_i$, pick $x_i$.
- When deriving conflict clauses, use the first unique implication points, and backtrack to the second highest (i.e., second largest) decision level in the derived conflict clause (which is level 0 if the clause is unit).
- When performing BCP, propagate implications in the increasing order of clause indices (i.e., if both $x_i$ and $x_j$ are unit clauses, and $i < j$, propagate $x_i$ first).

| Level | Decision | Implication Graph | Conflict Clause |
|---|---|---|---|
| $j$ | $literal@j$ | $\{\langle literal@k, literal@m, c_i \rangle, \ldots\}$ | $c_n : (literal \lor \ldots \lor literal)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

# 3 Variations on SAT (20 points)

Consider the following variations on the propositional satisfiability (SAT) problem discussed in Lecture 3:

**Partial Weighted MaxSAT** Given a CNF formula $\phi_H = \bigwedge_{c \in H} c$ corresponding to a set of *hard* clauses $H$, and a CNF formula $\phi_S = \bigwedge_{c \in S} c$ corresponding to a set of *soft* CNF clauses $S$ with weights $w : S \to \mathbb{Z}$, the Partial Weighted MaxSAT problem is to find an assignment $A$ to the problem variables that satisfies all the hard clauses and that maximizes the weight of the satisfied soft clauses. That is, $A \models \bigwedge_{c \in H} c$, and if we let $C = \{c \in S | A \models c\}$, then there is no $C' \subseteq S$ such that $H \cup C'$ is satisfiable and $\sum_{c' \in C'} w(c') > \sum_{c \in C} w(c)$.

**Pseudo-Boolean Optimization** Let $B$ be a set of *pseudo-boolean constraints* of the form $\sum a_{ij} x_j \geq b_i$, where $x_j$ is a variable over $\{0, 1\}$ and $a_{ij}, b_i, c_j$ are integer constants. The Pseudo-Boolean Optimization problem is to satisfy all constraints in $B$ while minimizing a linear function $\sum c_j \cdot x_j$.

7. (10 points) Explain how to encode a Partial Weighted MaxSAT problem $P$ as a Pseudo-Boolean Optimization problem $P'$ such that (1) $P'$ is satisfiable iff $P$ is satisfiable, and (2) a solution to $P$ can be extracted from a solution to $P'$. Show the result of your encoding (the optimization function and the pseudo-boolean constraints) on the following example, which uses the pair notation to associate weights with soft clauses:

$$\begin{array}{rcl} H & = & \{(x_1 \vee x_2 \vee \neg x_3), (\neg x_2 \vee x_3), (\neg x_1 \vee x_3)\} \\ S & = & \{\langle(\neg x_3), 6\rangle, \langle(x_1 \vee x_2), 3\rangle, \langle(x_1 \vee x_3), 2\rangle\} \end{array}$$

8. (10 points) Explain how to encode a Pseudo-Boolean Optimization $P$ as a Partial Weighted MaxSAT problem $P'$ such that (1) $P'$ is satisfiable iff $P$ is satisfiable, and (2) a solution to $P$ can be extracted from a solution to $P'$. Assume the existence of a function *toCNF* that takes as input a pseudo-boolean constraint $\sum a_{ij} x_j \geq b_i$ and encodes it as a boolean circuit in CNF form. Show the result of your encoding (the set of hard clauses, and the set of soft clauses with their weights) on the following example:

$$\begin{array}{ll} \text{minimize} & 4x_1 + 2x_2 + x_3 \\ \text{subject to} & 2x_1 + 3x_2 + 5x_3 \geq 5 \\ & -x_1 - x_2 \geq -1 \\ & x_1 + x_2 + x_3 \geq 2 \end{array}$$

# 4 Graph Coloring with SAT (40 points)

A graph is *k-colorable* if there is an assignment of $k$ colors to its vertices such that no two adjacent vertices have the same color. Deciding if such a coloring exists is a classic NP-complete problem with many practical applications, such as register allocation in compilers. In this problem, you will develop a CNF encoding for graph coloring and apply them to graphs from various application domains, including course scheduling, N-queens puzzles, and register allocation for real code.

A finite graph $G = \langle V, E \rangle$ consists of vertices $V = \{v_1, \ldots, v_n\}$ and edges $E = \{\langle v_{i_1}, w_{i_1} \rangle, \ldots, \langle v_{i_m}, w_{i_m} \rangle\}$. Given a set of $k$ colors $C = \{c_1, \ldots, c_k\}$, the *k-coloring* problem for $G$ is to assign a color $c \in C$ to each vertex $v \in V$ such that for every edge $\langle v, w \rangle \in E$, $\mathsf{color}(v) \neq \mathsf{color}(w)$.

9. (10 points) Show how to encode an instance of a $k$-coloring problem into a propositional formula $F$ that is satisfiable iff a $k$-coloring exists.
   (a) Describe a set of propositional constraints asserting that every vertex is colored. Use the notation $\mathsf{color}(v) = c$ to indicate that a vertex $v$ has the color $c$. Such an assertion is encodable as a single propositional variable $p_v^c$ (since the set of vertices and colors are both finite).
   (b) Describe a set of propositional constraints asserting that every vertex has at most one color.
   (c) Describe a set of propositional constraints asserting that no two adjacent vertices have the same color.
   (d) Identify a significant optimization in this encoding that reduces its size asymptotically. (**Hint:** Can any constraints be dropped? Why?)
   (e) Specify your constraints in CNF. For $|V|$ vertices, $|E|$ edges, and $k$ colors, how many variables and clauses does your encoding require?

10. (20 points) Implement the above encoding in Racket, using the provided solution skeleton. See the `README` file for instructions on obtaining solvers and the database of graph coloring problems. Your program should generate the encoding for a given graph (see `graph.rkt`), call a SAT solver on it (`solver.rkt`), and then decode the result into an assignment of colors to vertices (see `examples.rkt` and `k-coloring.rkt`).

    What is the minimum, maximum, and average solving time ("real" time if you are using Racket's `time` procedure) for easy and medium instances in the provided database of problems (see `problems.rkt`)? Can you solve any of the hard instances in 10 minutes or less?

11. (5 points) Describe a CNF encoding for $k$-coloring that uses $O(|V| \log k + |E| \log k)$ variables and clauses.

12. (5 points) Most modern SAT solvers support *incremental solving*—that is, obtaining a solution to a CNF, adding more constraints, obtaining another solution, and so on. Because the solver keeps (some) learned clauses between invocations, incremental solving is generally the fastest way to solve a series of related CNFs. How would you apply incremental solving to your encoding from Problem 10 to find the smallest number of colors needed to color a graph (i.e., its chromatic number)?